Das 68000-Paket

Benutzer-Handbuch HDT/RSU - 68000

Das 68000-Paket erhalten Sie bei:

Ing.-Büro Wilke · Postfach 1727 · D-5100 Aachen 1 · Tel. 0241/3 06 81 + 87 02 08

HDT-68000 Debugging Tool

RSU-68000 Runtime Simulator

Vers. 1.02

Bedienungsanleitung

Stand: 4.9.84 (3) HDT-HOM

Copyright

"Das 68000-Paket", bestehend aus Computer-Programmen und schriftlichen Unterlagen ist geistiges Eigentum des Lizenz-Inhabers, Hans-Jürgen Wilke, 5100 Aachen, Postfach 1727. Diesen Sachverhalt erkennen Händler und End-Abnehmer dieses Produktes an.

Mit Zahlung des 'Kaufpreises' entrichtet der End-Abnehmer eine Lizenzgebuhr, die ihn dazu berechtigt, diese Programme auf einem Computer ablaufen zu lassen und entsprechend dem hier beschiebenen Verwendungszweck zu benutzen (Einzel-Benutzerrecht). Dieses Recht ist nicht übertragbar, weitere Rechte bedurfen der schriftlichen Vereinbarung mit dem Lizenz-Inhaber.

Nicht gestattet sind insbesondere:

- das Kopieren des Produktes, oder Teilen hiervon, außer zum Zwecke der persönlichen Programmsicherung (Backup),
- die Weitergabe des Produktes oder Kopien hiervon, im Ganzen oder in Teilen,
- die Veränderung des Produktes oder überführung in eine andere Darstellungsform, also z.B.:
  das Listen, Disassemblieren, Decompilieren, übersetzen in andere Sprachen, die uberführung in ein anderes elektronisches oder nichtelektronisches Aufzeichnungs-Verfahren.
- Das gleichzeitige Benutzen dieses Produktes auf meheren Computer-Anlagen.

Der Händler erwirbt kein Benutzerrecht an diesem Produkt, vielmehr tritt er gegenüber dem Endbenutzer als Vermittler auf, der für seine Tätigkeit eine Vermittler-Provision (Handelsspanne) erhält.

Gewährleistungsausschluß

Der Lizenz-Inhaber behält sich vor, Änderungen an diesem Produkt vorzunehmen ohne die Verpflichtung diese irgendjemandem bekanntzugeben. Ferner ist jede Schadenersatz-Forderung an den Lizenz-Inhaber ausgeschlossen, falls im Zusammenhang mit diesem Produkt Kosten oder sonstige Schäden entstehen.

Inhaltsverzeichnis

1. Informationen zum MC 68000

1.1 Operandengröße

Der MC 68000 wird in der Literatur meistens als 16/32 Bit Prozessor bezeichnet. 16 Bit ist dabei die Breite des externen Datenbusses, 32 Bit die Breite der internen Register. Im Gegensatz zu 8 Bit Prozessoren können also 2 Bytes mit einem Buszyklus gelesen bzw. geschrieben werden und 32 Bit Daten mit einem Befehl verarbeitet werden. Die Operationsgröße wird bei der Programmierung mit angegeben. Dabei bedeuten

- .B = Byte Operand = 1 Byte = 8 Bit
- .W = Word Operand = 2 Byte = 16 Bit
- .L = Long Word Operand = 4 Bytes = 32 Bit

Es ist dabei zu beachten, daß das Lesen und Schreiben von Word und Long Word Operanden nur auf geraden Adressen möglich ist, da der Speicher des Prozessors in Wordbreite organisiert ist. Byte Operanden können dagegen auch ungerade Adressen haben. Bei Word Operanden steht dabei das MSB des Operanden auf der niederen Adresse, das LSB auf der höheren Adresse (umgekehrt wie es bei den meisten 8 Bit Systemen ist !).

1.2 Betriebsarten

Für den MC 68000 stehen zwei Betriebsarten zur Verfügung

- Supervisor (System) Mode
- User (Normal) Mode

Für beide Betriebsarten steht jeweils ein Stackpointer (A7) bereit, der gleichzeitig mit der jeweiligen Betriebsart durch ein Bit im Statusregister (S-Bit) ausgewählt wird. Die beiden Betriebsarten unterscheiden sich dadurch, daß im User Mode nicht alle Befehle verwendet werden dürfen. Normalerweise läuft ein Betriebssystem im Supervisor Mode und alle Anwenderprogramme im User Mode.

1.3 Register und Flags

Der MC 68000 besitzt im einzelnen folgende Register :

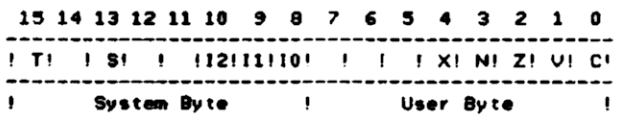
- 8 Datenregister zu 32 bit (D0 - D7)
- 7 Adressregister zu 32 bit (A0 - A6)
- 2 Stackpointer zu 32 bit (A7 = USP bzw. SSP)
- 1 Programcounter zu 32 bit (PC)
- 1 Statusregister zu 16 bit (SR)

Die Datenregister sind universell einsetzbare Register. Sie können für Operanden (Byte, Word oder Long Word), als Index Register oder aber auch als Zähler verwendet werden. Jedes Datenregister ist mit dem Akkumulator einer 8 Bit CPU zu vergleichen.

Adressregister können dagegen keine Byte Operanden verarbeiten, jedoch ebenso als Index register eingesetzt werden. Die Adressregistern ermöglichen vielfältige Adressierungsarten, in denen sich der eigentliche Vorteil des MC 68000 zeigt. Adressregister A7 dient als Stackpointer, wird aber genauso wie die übrigen Adressregister behandelt. Hinter Register A7 verbergen sich eigentlich zwei Register. In Abhängigkeit von Bit 13 (S-Bit) des Statusregisters (SR) wird durch A7 der User Stackpointer (USP) oder der Supervisor Stackpointer (SSP) adressiert.

Das Statusregister ist in zwei 8 Bit Bytes aufgeteilt:

- System Byte
- User Byte



- Statusregister-

Das System Byte (Bit 8 - 15) enthält eine 3 Bit Interruptmaske, die festlegt, welche Interruptlevels vom Prozessor verarbeitet werden, ein Bit um die Betriebsart des Prozessors zu bestimmen (S-Bit) und ein Bit um den Trace Mode einzuschalten (T-Bit).

Das User Byte enthalt die verschiedenen Flags :

- C = Carry

Das Carry Bit enthält den Übertrag aus dem obersten Bit (MSB), der durch arithmetische oder shif Operationen zustande gekommen ist.

- V = Overflow

Das Overflow Bit zeigt einen arithmetischen Überlauf an, d.h. es wird gesetzt, wenn das Ergebnis einer Operation größer als der maximal in einem Register darstellbare Zahlenbereich ist.

- Z = Zero

Das Zero Bit wird gesetzt, wenn das Ergebnis einer Operation gleich Null ist, ansonsten wird es zurückgesetzt.

- N = Negative

Das Negative Flag wird gleich dem MSB des Ergebnises einer Operation gesetzt.

- X = Extend

Das Extend Bit wird immer auf den selben Wert wie das Carry Bit gesetzt, sofern es durch die durchgeführte Operation beeinflußt wird.

1.4.5 Indirekte Adressierung durch ein Adressregister mit vorherigem Decrementieren (Address Register Indirect with Predecrement)

Operandenadresse = (An) - 1 (Byte Operation)
= (An) - 2 (Word Operation)
= (An) - 4 (Long Word Operation)

Beispiele :

```
CMPI.B    #CR,-(A0)     A0 := A0 - 1, dann
                        durch A0 angegebene
                        Speicherzelle mit Wert
                        des Labels CR vergleichen

TST       -(A5)         A5 := A5 - 1, dann
                        durch A5 angegebene
                        Speicherzelle auf 0
                        testen.
```

Register A5 habe den Wert 1000H. Zunächst wird A5 um 1 dekrementiert. A5 enthält dann den Wert 0FFFH. In obigem Beispiel wird dann der Inhalt der Speicherzelle 0FFFH auf 0 getestet.

1.4.6 Indirekte Adressierung durch ein Adressregister mit 16 Bit Displacement (Address Register Indirect with Displacement)

Operandenadresse = (An) + displacement

Beispiele :

```
ADD.W     $17F(A4),D2   Inhalt von A4 + $7F zu
                        Register D2 addieren
```

Register A4 habe den Wert 2000H. Die Adresse des Operanden ergibt sich durch Addition des Inhalts von Register A4 und des Displacements, in obigem Beispiel also : Operandenadresse = 2000H + 17FH = 217FH.

1.4.7 Indirekte Adressierung durch ein Adressregister mit 8 Bit Displacement und Indexregister (Address Register Indirect with Index)

Operandenadresse = (An) + displacement +
(Indexregister)

Beispiele :

```
CMP.W     2(A0,D1.W),D0
```

Register A0 habe den Wert 1000H, Register D1 den Wert 1234H. Die Operandenadresse errechnet sich dann zu : 1000H + 1234H + 2 = 2236H. Durch Angabe von .L bei dem Indexregister, wird der gesamte Inhalt des Index registers als Offset verwendet.

1.4.8 Kurze absolute Adressierung (Absolut Short)

Die Operandenadresse folgt unmittelbar als ein Word auf den Operationscode

Beispiele :

```
MOVE.B    4567H,D2      Byte aus der Speicherzelle
                        4567H nach D2 laden
```

1.4.9 Lange absolute Adressierung (Absolut Long)

Die Operandenadresse folgt unmittelbar als ein Long Word auf den Operationscode

```
MOVE.L    123456H,D2    Long Word aus Speicher
                        zelle 123456H nach D2
                        laden
```

1.4.10 Programmzähler mit Displacement (Programcounter with Displacement)

Operandenadresse = (PC) + displacement

Beispiele :

```
JMP       500(PC)
```

Der Programmzähler habe den Wert 1000H. Dann ergibt sich als Sprungziel in obigem Beispiel 1000H + 500

1.4.11 Programmzähler mit Displacement und Indexregister (Programcounter with Index)

Operandenadresse = (PC) + displacement + (Indexregister)

Beispiele :

```
ADD.L     $12(PC,A1),D6
```

Mit den Werten PC = 1000H und A1 = 2000H ergibt sich in obigem Beispiel die Operandenadresse zu 1000H + 2000H + 12H = 3012H

1.4.12 Unmittelbare Adressierung (Immediate Data)

Der Operand folgt unmittelbar auf den Operationscode

Beispiele :

```
OR.L      #$12345678,D1       logische ODER Ver-
                              knüpfung $12345678
                              ODER D1
```


1.5 Befehlssatz

Der Befehlssatz des MC 68000 umfasst 56 leistungsstarke Befehle. Zusammen mit den oben erwähnten Adressierungsmöglichkeiten lassen sich hiermit kompakte und gut strukturierte Programme erstellen.

In der Beschreibung des OPAL-68000 Assemblers finden Sie hierzu eine Beschreibung der einzelnen Befehle. Für ausführliche Informationen greifen Sie bitte auf die einschlägige Fachliteratur zurück.

Um das Verhalten eines Programmes mit einem STOP Befehl testen zu können, besteht die Möglichkeit nach Erreichen des STOP-Befehles einen RESET oder INTERRUPT auszulösen. Dies geschieht durch Tastatur-Eingabe von CTRL-R bzw. CTRL-I bzw. anderen Zeichen wie in den 'Installations-Hinweisen' angegeben.

2. HDT-68000 DEBUGGER

2.1 Einfuhrung

Der HDT-68000 Debugger simuliert auf Ihrem Rechner einen MC 68000 Prozessor und stellt die zum Testen von Programmen nötige Software zur Verfügung. Alle nicht von der Hardware abhangigen Befehle werden vollständig simuliert. Befehle die auf die Hardware zurückgreifen (RESET, STOP etc.) werden mit Einschränkungen simuliert. Bei der Behandlung von Ausnahmezuständen gibt es ebenfalls einige Einschränkungen. Diese werden weiter unten besprochen.

Starten des Debuggers mit : HDT68 <CR>

Nach Ausgabe der Copyright Meldung meldet sich der Debugger mit dem Zeichen '<' und ist bereit Kommandos anzunehmen.

Nach dem Starten sind alle Register, der Programmzähler und der User Stackpointer auf 0 gesetzt, der Supervisor Stackpointer auf 0400H. Das Statusregister enthällt den Wert 2700H, d.h. der Supervisor Mode ist eingeschaltet und die Interruptmaske steht auf der niedrigsten Stufe. Alle benötigten Vektoren sind so initialisiert, daß bei Fehlern die betreffenden Routinen des Debuggers anlaufen.

Hinweis : Ein Anwenderprogramm kann jederzeit durch Eingabe von CTRL-Q (bzw. einem anderen Zeichen, siehe 'Installations-Hinweise') angehalten werden. Nach Ausführung des aktuellen Befehles meldet sich der Debugger mit der Register-Anzeige. Programm-Abbruch innerhalb von Betriebssystem-Funktionsaufrufen ist mit CTRL-Q nicht möglich.

2.2 HDT-68000 Kommandos

Alle Simulator Kommandos, bei denen Zahlen als Parameter folgen, erwarten diese in hexadezimaler Schreibweise. Nach jedem Kommando ist die Return Taste zu betätigen. Die Kommandos können mit beliebig vielen Leerzeichen eingegeben werden. Werden mehr Zeichen als erlaubt eingegeben, so werden die letzten übernommen. Bei Eingabe von A0 = 123456789 wird dem Adressregister A0 der Wert 23456789 zugewiesen.

2.2.1 Adressregister setzen

* An = Hexwert
Das Adressregister An wird mit dem Wert HEXWERT geladen. Für n sind dabei die Werte 0 bis 6 zulässig. Register A7 wird mit den Befehlen SSP = bzw. USP = gesetzt.

2.2.2 Haltepunkt(e) löschen

* BC (Adresse)
Ohne Angabe einer Adresse werden alle bisher gesetzten Haltepunkte (Breakpoints) gelöscht. Mit Angabe einer Adresse wird der Haltepunkt auf dieser Adresse gelöscht.

2.2.3 Haltepunkte anzeigen

* BL
Alle Haltepunkte (Breakpoints) werden angezeigt. Der erste Wert gibt dabei die Adresse an, der zweite den Zähler.

2.2.4 Haltepunkte setzen

* BS Adresse (,Zähler)
An die eingegeben Adresse wird ein Haltepunkt gesetzt. Wird kein Wert für den Zähler eingegeben, so wird er auf 1 gesetzt. Die Programmausführung wird dann beim Erreichen dieser Adresse gestoppt. Durch Vorgabe eines Wertes n, wird erst nach dem n-ten Durchlauf gestoppt.

2.2.5 Datenregister setzen

* Dn = Hexwert
Das Datenregister Dn wird mit dem Wert HEXWERT geladen. Für n sind dabei die Werte 0 bis 7 zulässig.

2.2.6 Speicherbereich in HEX und ASCII ausgeben

* DU Startadresse, Endadresse
Der Speicherinhalt wird von STARTADRESSE bis ENDADRESSE ausgegeben. Falls der Speicherinhalt ein druckbares Asciizeichen enthällt, so wird dieses ebenfalls aus-gegeben. Nicht druckbare Zeichen werden durch einen Punkt dargestellt. Betätigen einer beliebigen Taste stoppt die Ausgabe.

2.2.7 Speicherbereich mit einer Konstanten füllen

* FI Startadresse,Endadresse,Byte
Der Speicherbereich von STARTADRESSE bis einschließlich ENDADRESSE wird mit dem Wert BYTE besetzt.

2.2.8 Programm starten

* GO (Adresse)
Das Kommando GO kann mit oder ohne Adressangabe eingegeben werden. Ohne Eingabe einer Adresse wird an der Adresse mit der Programmausführung begonnen, auf die der Programmzähler (PC) zeigt. Durch Eingabe einer Adresse startet die Programmausführung an dieser Adresse.

2.2.9 File laden

* LO Laufwerk:Name (,Startadresse)
Das Prögramm NAME wird von Laufwerk LAUFWERK ab Adresse STARTADRESSE geladen. Der Programmname darf keine Extension haben, da vom Debugger die Extension '.COD' automatisch hinzugefügt wird. Wird keine Adresse angegeen, so wird die am Fileanfang stehende genommen. (vom OPAL Assembler durch ORG vorgegeben, oder vom Debugger durch das SAVE Kommando eingetragen). Der Programmzähler wird auf die Startadresse des geladenen Programmes gesetzt.

2.2.10 Speicherbereich verschieben

* MO VON,BIS,NACH
Der Speicherbereich von Adresse VON bis Adresse BIS wird nach Adresse NACH verschoben.

2.2.11 Speicherinhalt verändern

* OP Adresse
Der Inhalt des Wortes in Adresse, Adresse+1 wird ausgegeben. Nach Eingabe eines '=' Zeichen kann das adressierte Wort verändert werden. Durch Betätigen der Returntaste wird das nächste Wort adressiert, durch '^' das vorherige (unterschiedliche Zeichen für verschiedene Computer-Typen, siehe 'Installations-Hinweise'). Durch Eingabe von Q wird der 'OPEN' Modus wieder verlassen.

2.2.12 Programmcounter setzen

* PC = Hexwert
Der Programmzähler (PC) wird auf Adresse Hexwert gesetzt.

2.2.13 Debugger verlassen

* QU
Programmende und zurück zum Betriebssystem.

2.2.14 Register anzeigen

* RE
Der Inhalt aller 68000 Register wird angezeigt. Register A7 wird als USP und SSP angezeigt.

2.2.15 File schreiben

* SA Laufwerk:Name,Startadresse,Länge
Der Speicherinhalt von STARTADRESSE bis ENDADRESSE (jeweils einschließlich) wird auf Laufwerk LAUFWERK unter dem Namen NAME abgespeichert.

2.2.16 Statusregister setzen

★ SR = Hexwert
Das Statusregister (SR) wird mit dem Wert HEXWERT geladen.

2.2.17 Supervisor Stackpointer setzen

★ SS = Hexwert
Der Supervisor Stack Pointer (SSP) wird mit dem Wert HEXWERT geladen.

2.2.18 Einzelschritt

★ ST (Adresse)
Der Single Step Mode wird eingeschaltet. Der Simulator beginnt mit der Programmausführung an der Adresse, auf die der Programmzähler zeigt, bzw an der eingegebenen. Nach der Ausführung eines Befehles werden alle Register angezeigt, und es wird eine Eingabe zur Fortsetzung des Programmes erwartet. Betätigen der Returntaste startet die Ausführung des nächsten Befehls, jede andere Taste beendet den Single Step Mode.

2.2.19 Trace

★ TR (Anzahl)
Beginn der Programmausfährung ab der Adresse, auf die der Programmzähler (PC) zeigt. Nach jedem Befehl werden alle Register angezeigt. Es wird die eingebene Anzahl Befehle durchgeführt. Betätigen irgendeiner Taste stoppt den Trace Modus.

2.2.20 User Stackpointer setzen

★ US = Hexwert
Der User Stack Pointer (USP) wird mit dem Wert HEXWERT geladen.

2.3 Behandlung von Ausnahmen (Execption Processing)

Die Behandlung von Ausnahmezuständen wird beim 68000 grundsätzlich über Vektoren durchgeführt. Hierzu ist im Speicher eine 1 Kbyte lange Vektortabelle vorhanden (Adresse 0000H - 03FFH). In diesem Speicherbereich sind die Vektoren als Longword (4 Bytes) abgelegt. Tritt ein Ausnahmezustand ein, so unterbricht der Prozessor die Bearbeitung des aktuellen Programmes und setzt die Programmausführung bei der Adresse fort, die er an der entsprechenden Stelle in der Vektortabelle findet. Vorher werden noch Informationen wie Programmzähler und Statusregister auf den Stack gerettet. Jeder Ausnahme ist eine Adresse zugeordnet, unter der der Prozessor den zugehörigen Vektor erwartet (Tabelle 1). Eine nicht initialisierte Vektortabelle kann daher leicht zum 'Abstürzen' des Programmes führen. Vermeiden Sie es daher, wenn nicht unbedingt erforderlich, die Vektoren in dieser Tabelle zu verändern.

Das Debug Programm schreibt in jeden Vektor die zugehörige Vektornummer ein. Tritt ein Ausnahmezustand ein, so wird getestet, ob der Vektor seine Nummer enthält. Falls dies zutrifft, führt das Debug Programm die Ausnahmebehandlung durch und gibt eine entsprechende Meldung aus. Wird jedoch ein anderer Wert gefunden, so wird eine Ausnahmebehandlung wie beim 'echten' 68000 durchgeführt. Unter der Adresse die im betreffenden Vektor vorgefunden wird, muß also ein sinnvolles 68000 Programm stehen. Hierdurch hat man also auch die Möglichkeit eigene Routinen für die Behandlung von Ausnahmezuständen zu benutzen.

Vektor Nr.	Adresse dez.	Adresse hex.	Zuordnung
0	0	0	Reset : Supervisor Stackpointer
1	4	4	Reset : Programmzähler
2	8	8	Bus Fehler
3	12	C	Adress Fehler
4	16	10	Illegaler Befehl
5	20	14	Division durch 0
6	24	18	CHK Befehl
7	28	1C	TRAPV Befehl
8	32	20	Privilegverletzung
9	36	24	Trace
10	40	28	Befehlscode Axxx Emulator
11	44	2C	Befehlscode Fxxx Emulator
12	48	30	reserviert
13	52	34	reserviert
14	56	38	reserviert
15	60	3C	nicht initialisierter Interrupt
16	64	40	
-	-	-	reserviert
23	92	5C	
24	96	60	falscher Interrupt
25	100	64	Auto-Interrupt Vektor für Ebene 1
26	104	68	Auto-Interrupt Vektor für Ebene 2
27	108	6C	Auto-Interrupt Vektor für Ebene 3
28	112	70	Auto-Interrupt Vektor für Ebene 4
29	116	74	Auto-Interrupt Vektor für Ebene 5
30	120	78	Auto-Interrupt Vektor für Ebene 6
31	124	7C	Auto-Interrupt Vektor für Ebene 7
32	128	80	Trap 0 Vektor
33	132	84	Trap 1 Vektor
34	136	88	Trap 2 Vektor
35	140	8C	Trap 3 Vektor
-	-	-	
47	188	BC	Trap 15 Vektor
48	192	C0	
-	-	-	reserviert
63	252	FC	
64	256	100	
-	-	-	Interruptvektoren für Anwender
255	1020	3FC	

- Tabelle 1 -
Ausnahme Vektoren

2.3.1 Reset

Nach einem durch die Hardware erzeugten Reset wird der Supervisorstackpointer aus Vektor 0, der Programmzähler aus Vektor 1 geladen. Diese Funktion wird vom Debugger nicht unterstützt.

2.3.2 Busfehler

Diese Ausnahme wird normalerweise auch von der Hardware ausgelöst. Vom Debugger wird ein Busfehler durch Zugriff auf eine nicht existierende Adresse ausgelöst.

Bei einem Busfehler werden außer Programmzähler und Statusregister noch zusätliche Informationen auf dem Stack abgelegt. Nach dem Statusregister folgt das Instruktionsregister, das den Opcode des letzten Befehls enthällt, anschließend die Zugriffsadresse als Longword und zuletzt ein Word, das Informationen darüber gibt, ob ein Lese oder Schreibzyklus stattgefunden hat, Code ausgeführt wurde und den Zustand der drei Funktionsleitungen angibt. Bis auf dieses letzte Statuswort werden vom Debugger ebenfalls alle Informationen auf dem Stack abgelegt, falls der ent-sprechende Vektor geändert wurde. An Stelle des Statuswortes werden 2 Bytes mit 0 auf den Stack abgelegt (Tabelle 2).

Hinweis: Ist der betreffende Vektor verändert, sodaß eine Anwenderroutine angesprungen wird und tritt in dieser Routine wiederum ein Busfehler auf (Stack-pointer zeigt auf nicht existierenden Rambereich), so wird in jedem Falle die Routine des Debuggers angesprungen.

niedere Adresse	! 0 ! 0 !	
	! Zugriffs- MSB !	
	! Adresse LSB !	
	! Instruktionsregister !	
	! Status Register !	
	! Programm- MSB !	
hohe Adresse	! zähler LSB !	

- Tabelle 2 -
Stack nach Bus- bzw. Adressfehler

2.3.3 Adressfehler

Ein Adress Fehler tritt auf, wenn der Prozessor einen Word- oder Longword-Zugriff auf eine ungerade Adresse durchführen will, oder aber der Programmzähler auf eine ungerade Adresse zeigt. Diese Funktion wird ebenfalls bis auf das Statuswort vom Debugger unterstützt.

2.3.4 Trace

Falls das Tracebit im Statusregister gesetzt ist, so wird nach jedem Befehl das Programm ausgeführt, auf das der Trace Vektor zeigt. Da diese Funktion hauptsächlich für Monitor Programme benötigt wird, wurde diese Funktion nicht implementiert. Tracen eines Programmes kann vom Debugger aus mit dem TR Befehl durchgeführt werden.

2.3.5 Befehlscode Axxx/Fxxx Emulation

Ein Opcode mit Bit 12-15 gleich 1010 oder 1111 führt zu diesen Vektoren (wird unterstützt).

2.3.6 Interrupt Vektoren

Da Interrupts von der verwendeten Hardware abhängen, werden diese Vektoren nicht unterstützt.

2.3.7 Trap Vektoren

Alle Trap Vektoren stehen frei zur Verfügung, ausgenommen Trap #0. Dieser Vektor wird verwendet um Betriebssystem aufrufe zu unterstützen.

3. RSU-68000 SIMULATOR

Der RSU-68000 stellt den Befehlssatz des MC 68000 Prozessors zur Verfügung. Als Eingangscode erwartet der Simulator 68000-Object-Code wie er z.B. vom OPAL-Assembler mit der Namens-erweiterung '.COD' erzeugt wird.

Starten des Simulators : RUN D:NAME

(siehe auch 'Installations-Hinweise')

NAME ist dabei ein File vom Typ .COD der vom OPAL Assembler, erzeugt wurde. D bezeichnet das Laufwerk, von dem der File geladen werden soll. Wird keine Extension angegeben, so wird .COD angenommen und als Default eingesetzt.

Die Programmausführung startet immer bei der durch das ORG Statement im Assembler angegebenen Adresse, bzw. bei der durch das SA Kommando erzeugten Adresse.

Falls ein Fehler in der Programmausführung auftritt, so werden die 68000 Register angezeigt, und ein Rücksprung ins Betriebssystem durchgeführt. Fehler können natürlich auch vom Benutzer durch Veränderung der entsprechenden Vektoren abgefangen werden.

Anhang A

Schnittstelle zum Betriebssystem

Runtime-Simulator wie auch der Debugger stellen eine Reihe von Betriebssystem-Funktionen zur Verfügung. Damit ist der Zugriff auf die Peripherie-Geräte des Computers von 68000-Maschinen-Programmen aus möglich.

Die verfügbaren Funktionen betreffen den Datenaustausch mit folgenden Geräten:

Bildschirm, Tastatur, Drucker,
Plattenspeicher (Diskette)

Alle Funktionen werden von einem 68000-Maschinen-Programm durch Ausführen des TRAP #0 Befehls angesprochen. Die Funktions-Auswahl geschieht durch Übergabe einer Funktions-Nummer im Register D3.B (Byte!), eventuelle Parameter werden in den Registern D0 und A0 übergeben.

Das Benutzer-Programm ist für die richtige Verwendung dieser Betriebssystem-Funktionen verantwortlich, da diese Funktionen nicht mehr im Simulator sondern im Betriebssystem des jeweiligen Computers ausgeführt werden. Besonders bei Disk-Zugriffen ist Vorsicht (und ev. ein Backup) geboten.

Funktion 0: Programm-Abbruch

Eingangs-Parameter: D3.B <---- 0
Ausgangs-Parameter: keine

Ausführen der Funktion 0 führt zu einem Abbruch des laufenden 68000-Maschinen-Programms. Anschließend befindet sich das System im gleichen Zustand, wie vor dem Starten des 68000-Programms.

Funktion 1: Tastatur-Eingabe

Eingangs-Parameter: D3.B <---- 1
Ausgangs-Paramater: D0.B <---- 1 Zeichen von der Tastatur

Funktion 1 wartet auf die Eingabe eines Zeichen auf der Tastatur, das Zeichen wird auf dem Bildschirm ausgegeben und in Register D0.B an das aufrufende Programm übergeben.

Funktion 2: Bildschirm-Ausgabe

Eingangs-Parameter: D3.B <---- 2
D0.B <---- 1 Zeichen für Ausgabe
Ausgangs-Parameter: keine

Das Zeichen in Register D0 wird an der aktuellen Cursor-Position ausgegeben.

Funktion 5: Drucker-Ausgabe

Eingangs-Parameter: D3.B <---- 5
D0.B <---- 1 Zeichen für Ausgabe
Ausgangs-Parameter: keine

Das Zeichen in Register D0 wird an der nächsten Druckposition auf dem Drucker ausgegeben.

Funktion 6: Tastatur-Status abfragen

Eingangs-Parameter: D3.B <---- 6
D0.B <---- $FF
Ausgangs-Parameter: D0.B <---- Status / Zeichen

Ohne auf eine Eingabe zu warten fragt die Funktion 6 den Status der Tastatur ab. Wurde zuvor ein Zeichen eingegeben, so wird es im Register D0 übergeben, wurde kein Zeichen eingegeben, enthält D0.B eine $00.

Funktion 9: String-Ausgabe auf Bildschirm

```
Eingangs-Parameter:   D3.B   <----   9
                      A0.L   <----   String-Adresse
Ausgangs-Parameter:   keine
```

Alle Zeichen ab der angegebenen Adresse bis zum Auffinden eines '$'-Zeichens werden auf dem Bildschrim ausgegeben.

Funktion 10: String-Eingabe von der Tastatur

```
Eingangs-Parameter:   D3.B   <----   10  ($0A)
                      A0.L   <----   Speicher-Adresse
Ausgangs-Parameter:   Eingabe-String liegt im Speicher vor.
```

A0.L gibt die Adresse eines Text-Speichers an. Nach Ausführen der Funktion 10 enthält dieser Speicher die eingegebenen Zeichen bis zum abschließenden <RETURN>.

Aufbau des Text-Speichers:

```
-------------------------------
! 1 ! 2 ! 3 ! 4 ! 5 ! .....
-------------------------------
```

In Position 1 wird die maximale Zahl der anzunehmenden Text-Zeichen festgelegt (1 - 255), Position 2 enthält die tatsächlich eingegebene Zeichen-Anzahl. Ab Position 3 steht der eingegebene String.

Funktion 15: öffne Disk-File

```
Eingangs-Parameter:   D3.B   <----   15  ($0F)
                      A0.L   <----   FCB-Adresse
Ausgangs-Parameter:   D0.B   <----   Fehler-/OK-Code
```

Funktion 15 öffnet einen Disk-File zum Lesen oder Schreiben. Das Register A0 enthält die Adresse des zugehörigen 'File-Control-Blocks'. Dieser FCB enthält die Drive- und und Namens-Angabe des Files. Aufbau des FCBs:

```
 1. Byte    Drive-Angabe:          1,2,3,4 ...
 2. Byte    8 Zeichen für
 .          Filename
 .               "
 9. Byte         "
10. Byte    3 Zeichen für Extent
11. Byte         "
12. Byte         "
13. Byte   ab dem 13.Byte mit $00
 .         gefüllt.
 .               "
36. Byte         "
```

Ist ein Filename kürzer als 8 Zeichen, wird bis zum 9.Byte mit dem Blank-Zeichen ($20) aufgefüllt, für den Extent gilt entsprechendes. Die Bytes 13-36 sind stets mit $00 vorzubesetzen. Zulässige Zeichen für Filenamen sind: GROSSBUCHSTABEN sowie je nach Computer-Typ und Betriebssystem: Ziffern und Satz-/Sonder-Zeichen.

Nach dem öffnen des Files mit der Funktion 15 darf der FCB nicht mehr verändert werden. Alle folgenden Lese- oder Schreib-Operationen auf diesen File, nehmen Bezug auf diesen FCB.

Als Ausgangs-Parameter wird in Register D0.B ein Fehler-/OK-Parameter übergeben: $00 = OK, Operation erfolgreich, jeder andere Wert: Fehler.

Funktion 16: File schließen

Eingangs-Parameter:	D3.B	<----	16 ($10)
	A0.L	<----	FCB-Adresse
Ausgangs-Parameter:	D0.B	<----	Fehler-/OK-Code

Schließt den in FCB spezifizierten File.

Funktion 19: File löschen

Eingangs-Parameter:	D3.B	<----	19 ($13)
	A0.L	<----	FCB-Adresse
Ausgangs-Parameter:	D0.B	<----	Fehler-/OK-Code

Funktion 19 löscht den im FCB spzifizierten File.

Funktion 20: File sequentiell lesen

Eingangs-Parameter:	D3.B	<----	20 ($14)
	A0.L	<----	FCB-Adresse
Ausgangs-Parameter:	D0.B	<----	Fehler-/OK-Code

Liest einen Record (128 Bytes) des spezifizierten Files in den zuletzt vereinbarten Disk-Puffer ein. D0.B enthält anschließend den Fehler-/OK-Code.

Funktion 21: File sequentiell schreiben

Eingangs-Parameter:	D3.B	<----	21 ($15)
	A0.L	<----	FCB-Adresse
Ausgangs-Parameter:	D0.B	<----	Fehler-/OK-Code

Schreibt einen Record (128 Bytes) des zuletzt vereinbarten Disk-Puffers auf den im FCB spzifizierten Disk-File. Ein File muß vor dem Beschreiben erfolgreich geöffnet worden sein. D0.B enthält anschließend den Fehler-/OK-Code.

Funktion 22: Neuen File erzeugen

Eingangs-Parameter:	D3.B	<----	22 ($16)
	A0.L	<----	FCB-Adresse
Ausgangs-Parameter:	D0.B	<----	Fehler-/OK-Code

Wie Funktion 15, jedoch nur für Schreiben, es wird ein neuer File des Namens wie im FCB angegeben angelegt. Ein eventuell schon existierender File gleichen Namens wird gelöscht !

Funktion 23: Filename ändern

Eingangs-Parameter:	D3.B	<----	23
	A0.L	<----	FCB-Adresse
Ausgangs-Parameter:	D0.B	<----	Fehler-/OK-Code

Ändert den Namen eines bestehenden Files um. Der FCB enthält nacheinander die beiden Filenamen:

Bytes 2 .. 12 enthalten den alten Filenamen,
Bytes 18 .. 28 enthalten den neuen Filenamen.

Die Drive-Angabe des alten Files legt den Disk-Drive fest, an Stelle der Drive-Angabe für den neuen Namen steht ein $00-Byte. Alle nicht benötigten Bytes des FCBs sind mit $00 zu besetzen:

Bytes 13 - 17, 29 - 36

Nach Ausführung enthält D0.B den Fehler-/OK-Code.

```
================================
 Funktion 26:   Setze Disk-Puffer
================================

 Eingangs-Parameter:    D3.B    <----   26  ($1A)
                        A0.L    <----   Puffer-Adresse
 Ausgangs-Parameter:    keine
```

Setzt den Disk-Puffer für Lese- und Schreib-Operationen auf eine bestimmte Adresse.

Anhang B

Es folgen einige Musterprogramme zum Ausprobieren. Alle Programme wurden mit dem OPAL-68000 Assembler übersetzt. Neben der Verwendung von 68000-Maschinen-Befehlen wird die Benutzung der Betriebssystem-Funktionen demonstriert.

```
OPAL-68000 Cross-Assembler  1.02 (C) - 1984  Wilke / IDA-Software   Seite 001
ADD64

                                TITLE   ESC,EA,'&ADD64',ESC,EA,'&'
                                LINIT   ESC,EA,'&'
                                LEXIT   ESC,EA,'0&'
                                LINE    131
                                PAGE    58
                                TOP     14

        0000001B        ESC:    EQU     1BH
        0000005B        EA:     EQU     5BH

                        ;
                        ; Name  : ADD64.M68
                        ; Typ   : OPAL 68000 Source Code
                        ; Stand : 07.09.84
                        ;
                        ; Hinweis : Initialisierung fuer Drucker LA 50

                        ; Addition von 64-bit BCD-Zahlen
                        ; zahl1 := zahl1 + zahl2


                                org     01000h

  001000  303A0028      N   start:  move.w  laenge(PC),D0   ;Laenge der BCD Zahlen in Bytes
  001004  41FA001C              lea.l   zahl1(PC),A0    ;Startadresse 1. Zahl
  001008  43FA001C              lea.l   zahl2(PC),A1
  00100C  D1C0                  adda.l  D0,A0           ;pointer auf niedrigste Stelle
  00100E  D3C0                  adda.l  D0,A1           ;der beiden Zahlen setzen
  001010  5340                  subq.w  #1,D0           ;dbf geht bis -1
  001012  44FC0000              move.w  #0,CCR          ;extend flag loeschen
  001016  C109              addi:   abcd.b  -(A1),-(A0)     ;BCD Addition
  001018  51C8FFFC              dbf     D0,addi

  00101C  163C0000              move.b  #0,D3
  001020  4E40                  trap    #0

                                even

  001022  12345678          zahl1:  dc.l    12345678H
  001026  00120034          zahl2:  dc.l    00120034H
  00102A  0004              laenge: dc.w    4
```



```
OPAL-68000 Cross-Assembler  1.02 (C) - 1984  Wilke / IDA-Software   Seite 002
ADD64

ADDI            00001016  label         EA              0000005B  equ         ESC             0000001B  equ
LAENGE          0000102A  label         START           00001000  N label     ZAHL1           00001022  label
ZAHL2           00001026  label
Keine Assemblier-Fehler
```

```
OPAL-68000 Cross-Assembler  1.02 (C) - 1984  Wilke / IDA-Software   Seite 001

MINMAX

                                TITLE   ESC,EA,'6,MINMAX',ESC,EA,'4w'
                                LIMIT   ESC,EA,'4w'
                                LEXIT   ESC,EA,'0w'
                                LINE    131
                                PAGE    58
                                TOP     14

        0000001B        ESC:    EQU     1BH
        0000005B        EA:     EQU     5BH


                        ;
                        ; Name  : MINMAX.M68
                        ; Typ   : OPAL 68000 Source Code
                        ; Stand : 07.09.84
                        ;       .
                        ; Hinweis : Initialisierung fuer Drucker LA 50


                        ; Bestimmung des groessten und kleinsten Wertes aus einer Liste von
                        ; Zahlen ohne Vorzeichen

                                org     01000h

  001000 41FA0024    N  start:  lea.l   tabelle(0),a0  ;Startadresse der Tabelle laden
  001004 4282               clr.l   d2              ;enthaelt spaeter Maximum
  001006 383CFFFF           move.w  #0ffffh,d4      ;enthaelt spaeter Minimum
  00100A 3018               move.w  (a0)+,d0        ;Laenge der tabelle uebernehmen
  00100C 5340               subq.w  #1,d0           ;DBF geht bis -1
  00100E 3218               loop:   move.w  (a0)+,d1        ;Wert uebernehmen
  001010 B441               cmp.w   d1,d2           ;groesser Maximum ?
  001012 6402               bcc.b   nein1
  001014 3401               move.w  d1,d2           ;neues Maximum uebernehmen
  001016 B841               nein1:  cmp.w   d1,d4           ;kleiner Minimum ?
  001018 6502               bcs.b   nein2
  00101A 3801               move.w  d1,d4           ;neues Minimum uebernehmen
  00101C 51C8FFF0           nein2:  dbf     d0,loop
  001020 163C0000           move.b  #0,d3           ;zurueck in Simulator
  001024 4E40               trap    #0

  001026                    tabelle:
  001026 0007               dc.w    7               ;7 Werte ueberpruefen
  001028 0402               dc.w    1234
  00102A 0929               dc.w    2345
  00102C 1DCF               dc.w    7631
  00102E 0009               dc.w    13
  001030 061E               dc.w    1566
  001032 07C8               dc.w    1992
  001034 0D05               dc.w    3333
```



```
OPAL-68000 Cross-Assembler  1.02 (C) - 1984  Wilke / IDA-Software   Seite 002

MINMAX

EA              0000005B  equ           ESC             0000001B  equ           LOOP            0000100E  label
NEIN1           00001016  label         NEIN2           0000101C  label         START           00001000  N label
TABELLE         00001026  label
Keine Assembler-Fehler
```

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wilke / IDA-Software Seite 001

PROG01

```
                                TITLE   ESC,EA,'&PROG01',ESC,EA,'&'
                                LIMIT   ESC,EA,'&'
                                LEXIT   ESC,EA,'&'
                                LINE    131
                                PAGE    58
                                TOP     14

        00000018        ESC:    EQU     1BH
        00000058        EA:     EQU     58H
                                LINE    131


                        ;
                        ; Name  : PROG01.M68
                        ; Typ   : OPAL 68000 Source Code
                        ; Stand : 01.09.84
                        ;
                        ; Hinweis : Benutzung von Systemfunktionen
                        ;          Initialisierung fuer Drucker LA 50


                        ;Programm zur Ausgabe eines Strings

                        ;Vereinbarungen

        0000000D        CR:             EQU     0DH             ;Carriage Return
        0000000A        LF:             EQU     0AH             ;Line Feed
        00000024        EOT:            EQU     '$'             ;Kennung Textende

        00000000        BDOS:           EQU     0               ;Funktionsaufruf
        00000009        PRINTSTRING:    EQU     9               ;Funktion 9

                                ORG     1000H                   ;Programm beginnt ab Adresse 1000H

001000  41FA000A       N       START:  LEA.L   TEXT(PC),A0       ;Startadresse des Strings
                                                                ;nach Register A0 laden
                                                                ;Adressierungsart PC relativ
001004  7609                    MOVEQ   #PRINTSTRING,D3         ;Funktionsnummer nach Register D3 laden
001006  4E40                    TRAP    #BDOS                   ;Systemaufruf --> Funktion ausfuehren

001008  4203                    CLR.B   D3                      ;LSB in Register D3 loeschen
                                                                ; --> Funktion 0 = zurueck in's System
00100A  4E40                    TRAP    #BDOS                   ;Funktion ausfuehren

                        ;Durch einen Systemaufruf ueber TRAP #0 mit der Funktionsnummer 0 wird das
                        ;User-Programm verlassen und das Betriebssystem angesprungen. Man kehrt also
                        ;entweder in den Debugger (Aufruf des Programmes im Debugger), oder aber in's
                        ;Betriebssystem zurueck.

00100C  000A                    TEXT:   DC.B    CR,LF           ;Cursor auf neue Zeile setzen
00100E  477574656E20            DC.B    'Guten Tag'             ;dieser Text wird auf der Console ausgegeben
001014  546167
```


OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wilke / IDA-Software Seite 002

PROG01

```
001017  24                      DC.B    EOT             ;Ende Kennung des Strings
```

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wilke / IDA-Software Seite 003

PROG01

BDOS	00000000	equ	CR	0000000D	equ	EA	0000005B	equ
EOT	00000024	equ	ESC	0000001B	equ	LF	0000000A	equ
PRINTSTRING	00000009	equ	START	00001000	N label	TEXT	0000100C	label

Keine Assemblier-Fehler

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wilke / IDA-Software Seite 001

PROG02

```
                        TITLE   ESC,EA,'6uPROG02',ESC,EA,'4u'
                        LINIT   ESC,EA,'4u'
                        LEXIT   ESC,EA,'0u'
                        LINE    131
                        PAGE    58
                        TOP     14

0000001B        ESC:    EQU     1BH
0000005B        EA:     EQU     5BH


                ;
                ; Name  : PROG02.M68
                ; Typ   : OPAL 68000 Source Code
                ; Stand : 01.09.84
                ;
                ; Hinweis : Benutzung von Systemfunktionen
                ;          Initialisierung fuer Drucker LA 50

                ;Programm zur Zeichen Ein- und Ausgabe

                ;Vereinbarungen

0000000D        CR:             EQU     0DH             ;Carriage Return
0000000A        LF:             EQU     0AH             ;Line Feed
00000024        EOT:            EQU     '$'             ;Kennung Textende
00000000        BDOS:           EQU     0               ;Funktionsaufrufe
00000000        EXIT:           EQU     0               ;Ruecksprung in's Betriebssystem
00000001        CONIN:          EQU     1               ;Zeichen von Console lesen
00000002        CONOUT:         EQU     2               ;Zeichen ausgeben
00000009        PRINTSTRING:    EQU     9

                        ORG     1000H                   ;Adresse fuer Programmbeginn

001000  612C    START:  BSR.B   CLEAR                   ;Bildschirm loeschen

001002  41FA00E8        LEA.L   MENUE(PC),A0            ;Startadresse Ausgabestring
001006  163C0009        MOVE.B  #PRINTSTRING,D3         ;Funktionsnummer 9 = Stringausgabe
00100A  4E40            TRAP    #BDOS                   ;String ausgeben

00100C  163C0001        MOVE.B  #CONIN,D3               ;Funktionsnummer 3 = Consolen Eingabe
001010  4E40            TRAP    #BDOS                   ;Zeichen einlesen

001012  0C000030        CMPI.B  #'0',D0                 ;Eingabe < '0' ?
001016  652A            BCS.B   FEHLER                  ;nicht erlaubt
001018  0C000034        CMPI.B  #'4',D0                 ;Eingabe > '4'
00101C  6424            BCC.B   FEHLER                  ;nicht erlaubt

00101E  04000030        SUBI.B  #30H,D0                 ;Umwandeln der ASCII Ziffer in Hex
001022  E540            ASL.W   #2,D0                   ;D0 ist wird als Zeiger auf das
                                                        ;gewuenschte Programm in der Tabelle benutzt
                                                        ;jede Adresse ist 4 Byte lang --> D0 * 4
001024  43FA006C        LEA.L   PRGTAB(PC),A1           ;Zeiger auf Tabelle mit Programmstartadressen
```

```
OPAL-68000 Cross-Assembler  1.02 (C) - 1984  Wilke / IDA-Software   Seite 002

PROG02

001028 24710000              MOVE.L  0(A1,D0.W),A2       ;Adresse aufgerufenes Programm
00102C 4ED2                  JMP     (A2)


                     ;CLEAR loescht den Bildschirm durch Ausgabe von 24 LF's

00102E 323C0017      CLEAR:  MOVE.W  #23,D1              ;D1 ist Zaehler
001032 163C0002      LOOP:   MOVE.B  #CONOUT,D3          ;Funktionsnummer 2 = Consolen Ausgabe
001036 103C000A              MOVE.B  #LF,D0              ;Ausgabezeichen
00103A 4E40                  TRAP    #BDOS               ;Zeichen ausgeben
00103C 51C9FFF4              DBF     D1,LOOP             ;Schleife solange wiederholen, bis D3 = 0FFFFH
                                                         ;d.h., bis Schleife 24 mal ausgefuehrt wurde.
001040 4E75                  RTS                         ;zurueck zum Hauptprogramm

                     ;Fehlerbehandlung

001042 41FA007E      FEHLER: LEA.L   MELDUNG($),A0       ;Adresse der Fehlermeldung
001046 163C0009              MOVE.B  #PRINTSTRING,D3     ;Funktion 9
00104A 4E40                  TRAP    #BDOS

00104C 323C01F4              MOVE.W  #500,D1             ;etwas warten
001050 51C9FFFE      WARTE:  DBF     D1,WARTE

001054 60AA                  BRA.B   START               ;und wieder zum Programmbeginn

001056 163C0000      ENDE:   MOVE.B  #EXIT,D3            ;Funktionsnummer
00105A 4E40                  TRAP    #BDOS

                     ;Hiernach ist das Programm zu Ende !!!

00105C 6128          PRG1:   BSR.B   TEXTAUS1
00105E 103C0031              MOVE.B  #'1',D0             ;ASCII '1' ausgeben
001062 163C0002              MOVE.B  #CONOUT,D3          ;Funktionsnummer
001066 4E40                  TRAP    #BDOS
001068 60EC                  BRA.B   ENDE

00106A 611A          PRG2:   BSR.B   TEXTAUS1
00106C 103C0032              MOVE.B  #'2',D0             ;ASCII '2' ausgeben
001070 163C0002              MOVE.B  #CONOUT,D3          ;Funktionsnummer
001074 4E40                  TRAP    #BDOS
001076 60DE                  BRA.B   ENDE

001078 610C          PRG3:   BSR.B   TEXTAUS1
00107A 103C0033              MOVE.B  #'3',D0             ;ASCII '3' ausgeben
00107E 163C0002              MOVE.B  #CONOUT,D3          ;Funktionsnummer
001082 4E40                  TRAP    #BDOS
001084 60D0                  BRA.B   ENDE


001086               TEXTAUS1:
001086 41FA001A              LEA.L   NUMMER($),A0
00108A 163C0009              MOVE.B  #PRINTSTRING,D3
00108E 4E40                  TRAP    #BDOS
001090 4E75                  RTS
```



```
OPAL-68000 Cross-Assembler  1.02 (C) - 1984  Wilke / IDA-Software   Seite 003

PROG02

                     ;Tabelle mit den Startadressen der einzelnen Programme

001092 00001056      PRGTAB: DC.L    ENDE
001096 0000105C              DC.L    PRG1
00109A 0000106A              DC.L    PRG2
00109E 00001078              DC.L    PRG3

                     ;Programmausgabe

0010A2 0D0A          NUMMER: DC.B    CR,LF
0010A4 417573667565          DC.B    'Ausfuehrung von Programm Nr. '
0010AA 6872756E6720
0010B0 766F6E20506F
0010B6 6F677261604D
0010BC 204E722E20
0010C1 24                    DC.B    EOT

                     ;Fehlermeldung

0010C2               MELDUNG:
0010C2 0D0A                  DC.B    CR,LF
0010C4 446965736520          DC.B    'Diese Eingabe ist nicht erlaubt'
0010CA 45696E676162
0010D0 652069737420
0010D6 6E6963687420
0010DC 65726C617562
0010E2 74
0010E3 24                    DC.B    EOT

                     ;String mit den Auswahlmoeglichkeiten

0010E4 0D0A          MENUE:  DC.B    CR,LF           ;Cursor auf neue Zeile
0010E6 417573776168          DC.B    'Auswahl'
0010EC 6C
0010ED 0D0A0A0A              DC.B    CR,LF,LF,LF
0010F1 3C313E2E2E2E          DC.B    '(1).....Programm Nr. 1'
0010F7 2E2E50726F67
0010FD 72616D6D204E
001103 722E2031
001107 0D0A0A                DC.B    CR,LF,LF
00110A 3C323E2E2E2E          DC.B    '(2).....Programm Nr. 2'
001110 2E2E50726F67
001116 72616D6D204E
00111C 722E2032
001120 0D0A0A                DC.B    CR,LF,LF
001123 3C333E2E2E2E          DC.B    '(3).....Programm Nr. 3'
001129 2E2E50726F67
00112F 72616D6D204E
001135 722E2033
001139 0D0A0A                DC.B    CR,LF,LF
00113C 3C303E2E2E2E          DC.B    '(0).....Programmende'
001142 2E2E50726F67
```

OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wilke / IDA-Software Seite 004

PROG02

```
001148 7261606D656E
00114E 6463
001150 0D0A0A0A0A           DC.B    CR,LF,LF,LF,LF
001155 426974746520         DC.B    'Bitte waehlen Sie aus : '
00115B 776165686C65
001161 6E2053696520
001167 617573203A20
00116D 24                   DC.B    EOT
```


OPAL-68000 Cross-Assembler 1.02 (C) - 1984 Wilke / IDA-Software Seite 005

PROG02

BDOS	00000000	equ	CLEAR	0000102E	label	CONIN	00000001	equ
CONOUT	00000002	equ	CR	0000000D	equ	EA	0000005B	equ
ENDE	00001056	label	EOT	00000024	equ	ESC	0000001B	equ
EXIT	00000000	equ	FEHLER	00001042	label	LF	0000000A	equ
LOOP	00001032	label	MELDUNG	000010C2	label	MENUE	000010E4	label
NUMMER	000010A2	label	PRG1	0000105C	label	PRG2	0000106A	label
PRG3	00001078	label	PRGTAB	00001092	label	PRINTSTRING	00000009	equ
START	00001000	label	TEXTAUS1	00001086	label	WARTE	00001050	label

Keine Assemblier-Fehler

OPAL-68000 Cross-Assembler 1.03 (C) - 1984 Wilke / IDA-Software Seite 001

DISKOUT.M68

```
                        TITLE   ESC,EA,'6wDISKOUT.M68',ESC,EA,'4w'

                        ;=================================================
                        ; Demonstration eines Disk-Outputs mit 68000 Simulator
                        ;=================================================

                        LINIT   ESC,EA,'4w'
                        LEXIT   ESC,EA,'0w'
                        LINE    131
                        PAGE    58
                        TOP     14

        0000001B        ESC:    EQU     1BH
        0000005B        EA:     EQU     5BH
        0000000D        CR:     EQU     0DH
        0000000A        LF:     EQU     0AH
        0000001A        eof:    equ     1ah
        00000024        STOP:   EQU     '$'

        00000000        programm_ende:  equ     0
        00000009        drucke_text:    equ     9
        00000010        close:          equ     16
        00000013        delete:         equ     19
        00000015        write:          equ     21
        00000016        make_file:      equ     22
        0000001A        setdma:         equ     26

        00000000        bdos:           equ     0
        000000FF        fehler:         equ     255
        00000000        kein_fehler:    equ     0
        00000080        record_laenge:  equ     80h
        00000001        LAUFWERK_A:     EQU     1

                                org     1000h
001000 41FA0190     N   start:  lea.l   fcb($),a0       ;FCB-Adresse
001004 163C0013             move.b  @delete,d3      ;alten File loeschen
001008 4E40                 trap    @bdos

00100A 163C0016             move.b  @make_file,d3   ;neuen File anlegen
00100E 4E40                 trap    @bdos

001010 B03C00FF             cmp.b   @fehler,d0
001014 6748                 beq.b   fehlerhaft

001016 123C0082             move.b  @&text_ende-text&/record_laenge ,d1
00101A 43FA0872             lea.l   text($),a1      ;DMA-Adresse in A1
00101E C149                 exg.l   a0,a1

001020 163C001A     loop:   move.b  @setdma,d3
001024 4E40                 trap    @bdos

001026 C149                 exg.l   a0,a1
001028 163C0015             move.b  @write,d3       ;Record schreiben
00102C 4E40                 trap    @bdos
```


OPAL-68000 Cross-Assembler 1.03 (C) - 1984 Wilke / IDA-Software Seite 002

DISKOUT.M68

```
00102E B03C0000             cmp.b   @kein_fehler,d0
001032 662A                 bne.b   fehlerhaft

001034 C149                 exg.l   a0,a1
001036 D1FC00000080         add.l   @record_laenge,a0
00103C 51C9FFE2             dbf     d1,loop

001040 C149                 exg.l   a0,a1
001042 163C0010             move.b  @close,d3       ;File schliessen
001046 4E40                 trap    @bdos

001048 B03C00FF             cmp.b   @fehler,d0
00104C 6710                 beq.b   fehlerhaft

                        ; hier alles ok
00104E 41FA0014             lea.l   ok_meldung(@),a0
001052 163C0009     ausgabe:move.b  @drucke_text,d3
001056 4E40                 trap    @bdos

001058 163C0000             move.b  @programm_ende,d3
00105C 4E40                 trap    @bdos

00105E                  fehlerhaft:
00105E 41FA0018             lea.l   fehler_meldung(@),a0
001062 60EE                 bra.b   ausgabe

001064                  ok_meldung:
001064 46696C652067         dc.b    'File geschrieben',cr,lf,stop
00106A 657363687269
001070 6562656E0D0A
001076 24

                            even
001078                  fehler_meldung:
001078 4469736B2053          dc.b    'Disk-Schreibfehler',cr,lf,stop
00107E 636872656962
001084 6665686C6572
00108A 0D0A24

                            even
00108E 0D0A             text:   dc.b    cr,lf
001090 566F6E206465         dc.b    'Von der Einsatzstelle der Ueberlegungen '
001096 72204569676E
00109C 61747A737465
0010A2 6C6C65206465
0010A8 722055656265
0010AE 726C656775E
0010B4 67656E20
0010B8 6861656E6774         dc.b    'haengt es naemlich',cr,lf
0010BE 206573206E61
0010C4 656D6C696368
0010CA 0D0A
0010CC 776573656E74         dc.b    'wesentlich ab, welche '
001002 6C6963682061
001008 622C2077656C
```

OPAL-68000 Cross-Assembler 1.03 (C) - 1984 Wilke / IDA-Software Seite 003

DISKOUT.M68

```
0010DE 63686520
0010E2 566F72617573             dc.b    'Voraussetzungen innerhalb des nach allen ',cr,lf
0010E8 7365747A756E
0010EE 67656E20696E
0010F4 6E657268616C
0010FA 622064657320
001100 6E6163682061
001106 6C6C656E200D
00110C 0A
00110D 53656974656E             dc.b    'Seiten unbegrenzten Bereiches moeglicher '
001113 20756E626567
001119 72656E7A7465
00111F 6E2042657265
001125 696368657320
00112B 6D6F65676C69
001131 6368657220
001136 50726F626C65             dc.b    'Probleme unbesehen',cr,lf
00113C 6D6520756E62
001142 65736568656E
001148 0D0A
00114A 75656265726E             dc.b    'uebernommen und welche '
001150 6F606D656E20
001156 756E64207765
00115C 6C63686520
001161 7A756D204765             dc.b    'zum Gegenstand der Besinnung gemacht',cr,lf
001167 765756E737461
00116D 6E6420646572
001173 2042657369GE
001179 6E756E672067
00117F 656D61636874
001185 0D0A
001187 77657264656E             dc.b    'werden.',cr,lf,eof
00118D 2E0D0A1A
001191                  text_ende:
                                even
                                fill    0
001192 01               fcb:    dc.b    laufwerk_a      ;File-Control-Block
001193 4445404F5445             dc.b    'DEMOTEXT'      ;--) Filename
001199 5854
001198 545854                   dc.b    'TXT'           ;--) File-Extent
00119E                          ds.b    24              ;--) Rest mit 00 vorbesetzen !
```


OPAL-68000 Cross-Assembler 1.03 (C) - 1984 Wilke / IDA-Software Seite 004

DISKOUT.M68

AUSGABE	00001052	label	BDOS	00000000	equ	CLOSE	00000010	equ
CR	0000000D	equ	DELETE	00000013	equ	DRUCKE_TEXT	00000009	equ
EA	00000058	equ	EOF	0000001A	equ	ESC	00000018	equ
FCB	00001192	label	FEHLER	000000FF	equ	FEHLERHAFT	0000105E	label
FEHLER_MELDU	00001078	label	KEIN_FEHLER	00000000	equ	LAUFWERK_A	00000001	equ
LF	0000000A	equ	LOOP	00001020	label	MAKE_FILE	00000016	equ
OK_MELDUNG	00001064	label	PROGRAMM_END	00000000	equ	RECORD_LAENG	00000080	equ
SETDMA	0000001A	equ	START	00001000	N label	STOP	00000024	equ
TEXT	0000108E	label	TEXT_ENDE	00001191	label	WRITE	00000015	equ

Keine Assemblier-Fehler

B e n u t z e r - K o m m e n t a r

Wir sind an Ihrer Meinung über dieses Produkt interessiert! Wenn Sie also Anregungen, Kritik oder Verbesserungvorschläge zu den Programmen oder der Dokumentation haben schreiben Sie uns.

Mir liegt folgende Programm-Version vor: _____

Bemerkungen zu Programm/Dokumentation: OPAL-68000

Bemerkungen zu Programm/Dokumentation: RSU-68000

Bemerkungen zu Programm/Dokumentation: HDT-68000

bitte senden an: Ing.-Büro Wilke, Postfach 1727, D-5100 Aachen 1

Stand dieser Übersicht: versi 1.06

Filenamen: HDT-68000 Debugger: 'HDT68.COM'
RSU-68000 Simulator: 'RUN.COM'
Simulator-Object-Code: 'FILENAME.COD' ('.COD' ist Default)

Starten des Debuggers:

A>HDT68 FILENAME(.COD) 'FILENAME(.COD)' wird geladen
oder: A>HDT68 kein File laden

Debugger-Kommandos:

Alle Zahlenwerte in HEX, Blanks überall zugelassen.

Register setzen:	Ax = nnnnnnnn Dx = nnnnnnnn SR = nnnnnnnn PC = nnnnnnnn USP = nnnnnnnn	
Register anzeigen:	RE	
Speicherzelle setzen:	OP	nur <RETURN>: nächste Adresse,
	OP adresse	'^' und <RETURN>: vorige Adresse, 'Q' und <RETURN>: Ende
Speicher anzeigen:	DU DU startadr DU startadr, endadr DU startadr S länge	
Speicher füllen:	FI startadr, endadr, byte	
Speicher verschieben:	MO startadr, endadr, zieladr	
Breakpoint setzen:	BS adresse BS adresse, zähler	
Breakpoint löschen:	BC	(alle)
	BC adresse	
Breakpoints anzeigen:	BL	(max. 50)
Programm starten:	GO	CTRL-Q: vorzeitiger Abbruch
	GO adresse	
Einzelschritt:	ST	jedes <RETURN>: weiterer Step
	ST startadresse	sonst: beenden
Trace:	TR anzahl	CTRL-Q: vorzeitiger Abbruch
File laden:	LO (d:)name(.ext)	
File schreiben:	SA (d:)name(.ext), start-1, länge-1, (start-2, länge-2, ...)	
Debugger verlassen:	QU	

```
    Dn      :          D3
    An      :          A1
   (An)     :         (A2)
   (An)+    :         (A3)+
   -(An)    :         -(A4)
   d(An)    :   OFFSET (A5)       :  -12X8 +SYMBOL(A2)
  d(An,Ri)  :   LABEL (A1,D2.W)   :  NEAR_BY +3 (A2,D2.L)
   Abs.W    :      @ LABEL.W
   Abs.L    :      @ LABEL.L      :       @LABEL
   d(PC)    :   DISPLACEMENT (@)
  d(PC,Ri)  : SMALL_DISPL (@, D6.L)
   Imm      : # [VIEL-3+K7] & SYMBOL

   CCR      -  Condition-Code Register (Flags)
    SR      -  Status-Register
   USP      -  User Stackpointer

 Reg-List   -  A2/A1/D1/D6/D3


Symbol-Definition:      ':'             SYMBOL:
  Symbol-Zeichen:  'A..Z', '0..9', '_'  NAME_1
Kommentar-Definition:     ';'           ; don't care ...
String-Definition:        '''           'String'
    Zahlenbasis 2:       'X2'           1010001111X2
    Zahlenbasis 8:       'X8'           77212601X8
   Zahlenbasis 10:       '-'            9124
   Zahlenbasis 16:       'H'            0A4FF5BDH
                        oder:           0A4FF5BD
```

Operatoren, Ausdrücke: * / ? = Multi, Divi, Rest
+ - = Plus, Minus
| & % = XOR, AND, OR (nach absteigender Priorität)

Klammerung zur Veränderung der Priortäten mit eckigen Klammern:

```
[ [ KLEIN + GROSS ] * 4 - WERT_2 ] & MASKE
```

Bei der Verwendung von String-Ausdrücken ist folgende Unterscheidung zu beachten:

'abcdefghi' kann als STRING oder als ZAHL benutzt werden:

```
        DC.B   'abcdefghi'        ; hier: String !
VAR_1:  EQU    'abcdefghi'        ; hier: Zahlenwert !
```

In diesen beiden Beispielen ist die Verwendung des Stringausdrucks als STRING bzw. als ZAHL eindeutig. Bei anderer Gelegenheit bedarf es einer entsprechenden Festlegung in der Programmzeile:

```
        DC.B   'abcdefghi'        ; OK, 9 ASCII-Zeichen
        DC.L   'abcdefghi' /4+7   ; Fehler !!
        DC.L   ['abcdefghi']/4+7  ; OK, arithm. Ausdr.
```

Bei allen Assembler-Anweisungen die Strings zulassen, ist deshalb eine entsprechende Kennzeichnung erforderlich, falls ein String als Zahl verwendet werden soll:

```
z.B:    0 + 'AB'
oder:   [ 'ABCDE' ]
```

Quell-Zeile: `LABEL:  OPCODE.X  op1,op2  ; Kommentar`
Listing-Zeile: `111 222222 333333333333 444 55555555555555555555555555555`

1 = Fehler-Feld, 0-3 Buchstaben kennzeichnen Fehler in Zeile
2 = Adress-Feld, 6 Hex-Ziffern
3 = Code-Feld, 0 - 12 Hex-Ziffern
4 = Flag-Feld, 0 - 3 Flags: M/-/R/C
M-Flag: Nicht benutztes Symbol
\- Flag: Symbol-REDEF
R-Flag: Range-Flag, kleinere Size möglich
C-Flag: INCLUDE ist aktiv
5 = Quelltext-Feld

Aufbau Symbol-Tabelle: SYMBOL_NAME, (Symbol-Wert), Flags, Art der Definition

SYMBOL_NAME: ---> 1..12 Zeichen (A..Z, 0..9, _)
(Symbol-Wert): ---> 8 Stellen hexadezimal (LONG)
Flags: ---> M = Fehler, Mehrfach-Definition
N = Nicht benutztes Symbol
\- = redefiniertes Symbol
Art der Definition: ---> equ, redef, label, input

Fehlermeldungen:

A =	Argument-Fehler, unzul. ADR-Mode	N =	Numerischer Fehler
D =	Disk-Fehler	O =	Opcode-Fehler
I =	Illegales Zeichen	P =	Phasen-Fehler
L =	LABEL-Fehler	R =	Bereichs-uberschreitung
M =	Mehrfach-Definition bzw. Benutzung eines solchen Symbols	S =	SIZE-Fehler
		U =	Undefiniertes SYMBOL
		X =	sonst. Fehler

Patches: Zur Anpassung an verschiedene Terminal-Typen können einzelne Zeichen des Assembler-Source-Codes durch andere substituiert werden:

```
START + 25H:  DB    Anzahl der Zeichen-Substitutionen
              DB    1. neues Zeichen
              DB    1. ursprüngliches Zeichen
              DB    2. neues Zeichen
              .. etc
```

Beispiel: Ein Terminal besitze keine eckigen Klammern, wie sie für die Arithmetik des Assemblers benötigt werden. Abhilfe: '[' und ']' werden ersetzt durch '(' und ')'. Erforderlicher Patch:

```
START + 25H:  DB    2       ; 2 Zeichen substituieren!
              DB    '('     ; neues Zeichen (1)
              DB    '['     ; ursprüngliches Zeichen (1)
              DB    ')'     ; neues Zeichen (2)
              DB    ']'     ; ursprüngliches Zeichen (2)
```

START = 103H in CP/M-80 Systemen

Folgende Patches sind in meinem CPAL-Assembler vorgenommen:

ursprüngliches Zeichen:	neues Zeichen:	Bedeutung des Zeichens:

Stand dieser Übersicht: vom: 1.06

Assembler-Aufruf: A)OPAL FILENAME(.EXT (/Switch-1 (/Switch-2 ..)))
Beispiel: A)OPAL B:HALLO /PP/EM
Abbruch des Assemblerlaufs: CTRL-C Eingabe (mit anschließender Bestätigung)

Command-Line Switches:

			Defaults:
/Px	x = A-M	--> Drive für Listing-File	source-dr'
	x = N	--> kein Listing	
	x = P	--> Listing an Printer	
	x = X	--> Listing an Console	
	x = Y	--> Listing an AUX-Output	
/F		--> nur fehlerhafte Zeilen listen	
/Ox	x = A-M	--> Drive für Object-File	'source-dr'
	x = N	--> kein Object-File erzeugen	
/Ex	x = A-M	--> Drive für EPROM-Daten	'source-dr'
	x = N	--> kein EPROM-Daten File	

File-Namen im Rahmen der Betriebs-System-Vorgaben frei wählbar, für Namens-Erweiterungen gilt:

.M68	=	68000-Source Code (Default)
.LST	=	Listing-File (immer)
.COD	=	Object-File (immer)
.BIN	=	Binär-File f. EPROM (immer)

Pseudo-Opcodes:

			Default:
LIST	- ohne Arg.:	schaltet Listing ein	ein
XLIST	- ohne Arg.:	schaltet Listing aus	-
PAGE	- ohne Arg.:	neue Seite beginnen	-
	mit Arg.:	setzt Seitenlänge	68
TOP	- mit Arg.:	setzt Anzahl Leerzeilen zischen 2 Seiten	4
LINE	- mit Arg.:	setzt Zeilenlänge	79
LINIT	- mit Arg.:	Drucker-Init-Sequenz	-
LEXIT	- mit Arg.:	Drucker-Exit-Sequenz	-
PUNCH	- ohne Arg.:	drucke Maschinen-Code aus	ein
XPUNCH	- ohne Arg.:	unterdrücke Ausdruck des Masch-Codes	-
TITLE	- mit Arg.:	Titel-Zeile für Listing	-
FLAG	- ohne Arg.:	zeige Flags im Listing an	ein
XFLAG	- ohne Arg.:	unterdrücke Listing-Flags	-
DC.X	- mit Arg.:	definiere Konstanten	-
		.X = opt. Size-Angabe	
DS.X	- mit Arg.:	definiere Speicherbereich	-
		.X = opt. Size-Angabe	
FILL	- mit Arg.:	setzt Füll-Zeichen für 'DS.X'	00H
EVEN	- ohne Arg.:	PC auf nächste gerade ADR stellen	-
EQU	- mit Arg.:	Symbol-Definition	-
ORG	- mit Arg.:	Adress-Definition für Programm-Segment	-
SIZE.X	- ohne Arg.:	setzt Default-Size	LONG
PRINT	- mit Arg.:	drucke Argument auf Console (Pass-1)	-
INPUT	- ohne Arg.:	hole Wert von Benutzer (Pass-1)	-
IFE	- mit Arg.:	Beginn conditional-Assembly	-
IFN	- mit Arg.:	Beginn conditional-Assembly	-
IFP	- mit Arg.:	Beginn conditional-Assembly	-
IFM	- mit Arg.:	Beginn conditional-Assembly	-
ENDIF	- ohne Arg.:	beendet conditional-Assembly	-
INCLUDE	- mit Arg.:	liest Text-File ein	-
REDEF	- mit Arg.:	Redefinition eines Symbols	-

4C-68000 Opcodes:

ABCD.X	op1,op2	- Add dec. w Extend
ADD.X	op1,op2	- Add Binary
ADDA.X	op1,op2	- Add Address
ADDI.X	op1,op2	- Add Immediate
ADDQ.X	op1,op2	- Add Quick
ADDX.X	op1,op2	- Add Extended
AND.X	op1,op2	- Logical AND
ANDI.X	op1,op2	- Logic AND Imm. (+)
ASL.X	op1(,op2)	- Aritha. Shift L.
ASR.X	op1(,op2)	- Aritha. Shift R.
Bcc.X	op1	- Branch Cond.
BCHG.X	op1,op2	- Test Bit/Change
BCLR.X	op1,op2	- Test Bit/Clear
BRA.X	op1	- Branch
BSET.X	op1,op2	- Test Bit/Set
BSR.X	op1	- Branch Subr.
BTST.X	op1,op2	- Test Bit
CHK.W	op1,op2	- Check Reg.v.Bounds
CLR.X	op1	- Clear an Operand
CMP.X	op1,op2	- Compare
CMPA.X	op1,op2	- Compare Address
CMPI.X	op1,op2	- Compare Immediate
CMPM.X	op1,op2	- Compare Memory
DBcc.W	op1,op2	- Test, Dec, Branch
DIVS.W	op1,op2	- Divide with Sign
DIVU.W	op1,op2	- Divide Unsigned
EOR.X	op1,op2	- Logical XOR
EORI.X	op1,op2	- Logic XOR Imm. (+)
EXG.L	op1,op2	- Exchange Regs
EXT.X	op1	- Sign Extend
JMP	op1	- Jump
JSR	op1	- Jump to Subr.
LEA.L	op1,op2	- Load Eff. Addr
LINK	op1,op2	- Link/Alloc Stack
LSL.X	op1,op2	- Log. Shift Left
LSR.X	op1,op2	- Log. Shift Right

MOVE.X	op1,op2	- Move Data (+)
MOVEA.X	op1,op2	- Move Address
MOVEM.X	op1,op2	- Move Mult. Regs
MOVEP.X	op1,op2	- Move Peripheral
MOVEQ.L	op1,op2	- Move Quick
MULS.W	op1,op2	- Multiply, Sign
MULU.W	op1,op2	- Multiply Unsigned
NBCD.B	op1,op2	- Negate Dec, Extend
NEG.X	op1	- Negate
NEGX.X	op1	- Negate with Extend
NOP		- No Operation
NOT.X	op1	- Logical Not
OR.X	op1,op2	- Logical Or
ORI.X	op1,op2	- Logic Or Imm. (+)
PEA.L	op1	- Push Eff. Addr
RESET		- Reset Ext-Dev: (*)
ROL.X	op1(,op2)	- Rotate Left
ROR.X	op1(,op2)	- Rotate Right
ROXL.X	op1(,op1)	- Rotate Left, Extend
ROXR.X	op1(,op2)	- Rotate Right,Extend
RTE		- Ret Exception (*)
RTR		- Ret and Restore CCR
RTS		- Ret from Subroutine
SBCD.B	op1,op2	- Sub Deci w. Extend
Scc.B	op1	- Set Conditionaly
STOP	op1	- Load SR, Stop (*)
SUB.X	op1,op2	- Subtract Binary
SUBA.X	op1,op2	- Subtract Address
SUBI.X	op1,op2	- Subtract Immediate
SUBQ.X	op1,op2	- Subtract Quick
SUBX.X	op1,op2	- Subtract w Extend
SWAP.W	op1	- Swap Reg-Halves
TAS.B	op1	- Test/Set Operand
TRAP	op1	- Trap
TRAPV		- Trap on Overflow
TST.X	op1	- Test an Operand
UNLK	op1	- Unlink

(*) = privilegierter Befehl, (+) = privilegierter Befehl, sofern SR oder USP angesprochen.

Die Opcodes: ADDA, CMPA, CMPI, MOVEA, SUBA und SUBI
können auch durch: ADD, CMP, MOVE, SUB ... abgekürzt werden.

Bitte beachten Sie: Ein OPAL-Source Text benötigt mindestens eine ORG-Anweisung!

IDA Ingenieurbüro für Datenverarbeitung

Ausnahme-Behandlung: Sofern der entsprechende Ausnahme-Vektor nicht verändert wurde, erfolgt Bearbeitung durch den Debugger. Ansonsten wird die zugehörige USER-Routine ausgeführt.

Ausnahme Vektoren:

Vektor Nr.	Adresse dez.	Adresse hex.	Zuordnung
0	0	0	Reset : Supervisor Stackpointer
1	4	4	Reset : Programmzähler
2	8	8	Bus Fehler
3	12	C	Adress Fehler
4	16	10	Illegaler Befehl
5	20	14	Division durch 0
6	24	18	CHK Befehl
7	28	1C	TRAPV Befehl
8	32	20	Privilegverletzung
9	36	24	Trace
10	40	28	Befehlscode Axxx Emulator
11	44	2C	Befehlscode Fxxx Emulator
12	48	30	reserviert
13	52	34	reserviert
14	56	38	reserviert
15	60	3C	nicht initialisierter Interrupt
16	64	40	
-	-	-	reserviert
23	92	5C	
24	96	60	falscher Interrupt
25	100	64	Auto-Interrupt Vektor für Ebene 1
26	104	68	Auto-Interrupt Vektor für Ebene 2
27	108	6C	Auto-Interrupt Vektor für Ebene 3
28	112	70	Auto-Interrupt Vektor für Ebene 4
29	116	74	Auto-Interrupt Vektor für Ebene 5
30	120	78	Auto-Interrupt Vektor für Ebene 6
31	124	7C	Auto-Interrupt Vektor für Ebene 7
32	128	80	Trap 0 Vektor
33	132	84	Trap 1 Vektor
34	136	88	Trap 2 Vektor
35	140	8C	Trap 3 Vektor
-	-	-	
47	188	BC	Trap 15 Vektor
48	192	C0	
-	-	-	reserviert
63	252	FC	
64	256	100	
-	-	-	Interruptvektoren für Anwender
255	1020	3FC	